

Universidad de las Fuerzas Armadas **ESPE**



Fundamentos de la programación

Algoritmos de búsqueda en C++

Introducción a los Algoritmos de Búsqueda en C++

- Los algoritmos de búsqueda son técnicas fundamentales en ciencias de la computación y son utilizados para encontrar un elemento específico dentro de una colección de datos. Estos algoritmos son esenciales para resolver problemas de recuperación de información, procesamiento de datos y optimización en una amplia gama de aplicaciones informáticas.

Importancia de los Algoritmos de Búsqueda

- Los algoritmos de búsqueda son cruciales debido a su aplicación en:

1. **Bases de Datos y Sistemas de Gestión de Información:** Permiten la recuperación eficiente de datos almacenados, facilitando consultas rápidas y precisas.
2. **Algoritmos de Ordenamiento:** Muchos algoritmos de ordenamiento requieren búsquedas internas para operar eficientemente.
3. **Inteligencia Artificial y Minería de Datos:** Se utilizan para buscar patrones, coincidencias o relaciones significativas en grandes conjuntos de datos.
4. **Algoritmos de Camino más Corto:** En problemas de redes y grafos, los algoritmos de búsqueda son fundamentales para encontrar la ruta más corta entre dos nodos.

Tipos Comunes de Algoritmos de Búsqueda

- **Búsqueda Lineal (Linear Search):** Es el método más simple donde cada elemento se compara secuencialmente hasta encontrar el objetivo o recorrer todo el conjunto de datos.
- **Búsqueda Binaria (Binary Search):** Requiere que los datos estén ordenados y divide repetidamente la lista en mitades, eliminando la mitad en la que no puede estar el elemento hasta encontrarlo o determinar que no está presente.
- **Búsqueda por Interpolación (Interpolation Search):** Similar a la búsqueda binaria, pero más eficiente en datos uniformemente distribuidos, calculando posiciones más probables basadas en la distribución de los datos.

- **Búsqueda por Saltos (Jump Search):** También conocida como búsqueda en bloques, divide el conjunto de datos en bloques de tamaño fijo y realiza una búsqueda lineal en el bloque donde puede estar el elemento.
- **Búsqueda Exponencial (Exponential Search):** Especialmente eficiente para arreglos ordenados y permite encontrar un rango donde puede estar el elemento antes de realizar una búsqueda binaria.

Búsqueda por Saltos (Jump Search):

- La **Búsqueda por Saltos (Jump Search)** es un algoritmo de búsqueda para encontrar un elemento en una lista ordenada. A diferencia de la búsqueda lineal (que revisa cada elemento uno por uno) y la búsqueda binaria (que divide repetidamente la lista a la mitad), la búsqueda por saltos da "saltos" de un tamaño fijo a través de la lista, haciendo la búsqueda más eficiente.

Aquí hay una explicación sencilla de cómo funciona:

1. **Determinar el tamaño del salto**: Usualmente, se elige como la raíz cuadrada del tamaño de la lista. Esto balancea el número de saltos y el número de comprobaciones individuales.
2. **Saltar en la lista**: Saltar a través de la lista en incrementos del tamaño del salto hasta que se encuentra un valor mayor o igual al valor buscado.
3. **Búsqueda lineal**: Realizar una búsqueda lineal en el bloque donde se detuvo el salto.

Supongamos que tenemos una lista ordenada y queremos encontrar un valor específico:

```
#include <iostream> // Biblioteca estándar para la entrada y salida
#include <cmath> // Biblioteca matemática estándar que incluye la función sqrt

// Función de búsqueda por saltos (Jump Search)
int jumpSearch(int arr[], int size, int value) {
    int step = std::sqrt(size);  // Determinar el tamaño del salto basado en la raíz cuadrada del tamaño del arreglo
    int prev = 0; // Inicializar el índice anterior a 0

    // Saltar a través del arreglo en pasos de 'step'
    while (arr[std::min(step, size) - 1] < value) { // Mientras el valor en el índice 'step-1' sea menor que 'value'
        prev = step; // Actualizar el índice anterior al valor actual de 'step'
        step += std::sqrt(size); // Incrementar 'step' en la raíz cuadrada del tamaño del arreglo
        if (prev >= size) // Si el índice anterior es mayor o igual al tamaño del arreglo
            return -1; // El valor no se encuentra en el arreglo
    }

    // Búsqueda lineal en el bloque identificado
    for (int i = prev; i < std::min(step, size); i++) { // Desde el índice 'prev' hasta el mínimo entre 'step' y 'size'
        if (arr[i] == value) // Si el valor en el índice 'i' es igual a 'value'
            return i; // Devolver el índice 'i' donde se encontró el valor
    }

    return -1; // El valor no se encuentra en el arreglo
}

// Función principal
int main() {
    int arr[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}; // Definir un arreglo de enteros
    int size = sizeof(arr) / sizeof(arr[0]); // Calcular el tamaño del arreglo
    int value = 13; // Definir el valor a buscar

    int index = jumpSearch(arr, size, value); // Llamar a la función de búsqueda por saltos

    if (index != -1) { // Si el valor fue encontrado (índice diferente de -1)
        std::cout << "Elemento encontrado en el índice " << index << std::endl; // Imprimir el índice donde se encontró el valor
    } else { // Si el valor no fue encontrado (índice igual a -1)
        std::cout << "Elemento no encontrado" << std::endl; // Imprimir que el valor no fue encontrado
    }

    return 0; // Finalizar el programa
}
```